

Definitions Template

B/defnt

Purpose

Miscellaneous constant definitions, usually providing symbolic names for otherwise inscrutable numbers, which are used throughout the template layer.

B/defnt. §1 VM Target Constants; §2 Wordsize-Dependent Definitions; §3 Z-Machine Definitions; §4 Glulx Definitions; §5 Powers of Two; §6 Text Styles; §7 Colour Numbers; §8 Window Numbers; §9 Paragraphing Flags; §10 Descriptors in the Language of Play; §11 Run-Time Problem Numbers; §12 Template Activities; §13 Template Rulebooks; §14 Data Type IDs; §15 Parser Error Numbers; §16 Scope Searching Reasons; §17 Token Types; §18 GPR Return Values; §19 List Styles; §20 Lengths Of Time; §21 Empty Text; §22 Empty Table; §23 Empty Rulebook; §24 Empty Set; §25 Score and Rankings Table; §26 Template Attributes; §27 Template Properties; §28 Loss of Life; §29 Action Count; §30 Fake Actions

§1. VM Target Constants. Inform can compile story files for four different virtual machines (or VMs): Z-machine versions 5, 6 and 8, and Glulx.

When compiling to Glulx, the I6 compiler predefines the constant `TARGET_GLULX` and also sets `WORDSIZE` to 4; but when compiling to Z, we shouldn't assume it has this modern habit, so we simulate the same effect.

```
#ifndef WORDSIZE; ! compiling with Z-code only compiler
Constant TARGET_ZCODE;
Constant WORDSIZE 2;
#endif;
```

§2. Wordsize-Dependent Definitions. The old I6 library used to confuse Z-vs-G with 16-vs-32-bit, but we try to separate these distinctions here, even though at present the Z-machine is our only 16-bit target and Glulx our only 32-bit one. The `WORDSIZE` constant is the word size in bytes, so is the multiplier between `->` and `-->` offsets in I6 pointer syntax.

- (1) `NULL` is used, as in C, to represent a null value or pointer. In C, this is conventionally 0, but here it is the maximum unsigned value which can be stored, pointing to the topmost byte in the directly addressable memory map; this means it is also `-1` when regarded as a signed twos-complement integer, but we write it as an unsigned hexadecimal address for clarity's sake.
- (2) `WORD_HIGHBIT` is the most significant bit in the VM's data word.
- (3) `IMPROBABLE_VALUE` is one which is *unlikely but still possible* to be a genuine I7 value. The efficiency of some algorithms depends on how well chosen this is: they would ran badly if we chose 1, for instance.
- (4) `MAX_POSITIVE_NUMBER` is the largest representable positive (signed) integer, in twos-complement form.
- (5) `REPARSE_CODE` is a magic value used in the I6 library's parser to signal that some code which ought to have been parsing a command has in fact rewritten it, so that the whole command must be re-parsed afresh. (Returning this value is like throwing an exception in a language like Java, though we don't implement it that way.) A comment in the 6/11 library reads: "The parser rather gunkily adds addresses to `REPARSE_CODE` for some purposes. And expects the result to be greater than `REPARSE_CODE` (signed comparison). So Glulx Inform is limited to a single gigabyte of storage, for the moment." Guilty as charged, but the gigabyte story file is a remote prospect for now: even megabyte story files are off the horizon. Anyway, it's this comparison issue which means we need a different value for each possible word size.

```
#iftrue (WORDSIZE == 2);
Constant NULL = $ffff;
Constant WORD_HIGHBIT = $8000;
Constant WORD_NEXTTOHIGHBIT = $4000;
```

```

Constant IMPROBABLE_VALUE = $7fe3;
Constant MAX_POSITIVE_NUMBER 32767;
Constant MIN_NEGATIVE_NUMBER -32768;
Constant REPARSE_CODE = 10000;
#Endif;

#Iftrue (WORDSIZE == 4);
Constant NULL = $fffffff;
Constant WORD_HIGHBIT = $80000000;
Constant WORD_NEXTTOHIGHBIT = $40000000;
Constant IMPROBABLE_VALUE = $deadce11;
Constant MAX_POSITIVE_NUMBER 2147483647;
Constant MIN_NEGATIVE_NUMBER -2147483648;
Constant REPARSE_CODE = $40000000;
#Endif;

```

§3. Z-Machine Definitions. The Z-machine contains certain special constants and variables at fixed position in its “header”; the addresses of these are given below. See *The Z-Machine Standards Document*, version 1.0, for details.

INDIV_PROP_START is the lowest number of any “individual property”, an I6 internal quantity defined by the compiler when the target is Glulx but not for Z.

```

#Ifdef TARGET_ZCODE;
Global max_z_object;
Constant INDIV_PROP_START 64;
! Offsets into Z-machine header:
Constant HDR_ZCODEVERSION      = $00;    ! byte
Constant HDR_TERPFLAGS        = $01;    ! byte
Constant HDR_GAMERELEASE      = $02;    ! word
Constant HDR_HIGHMEMORY       = $04;    ! word
Constant HDR_INITIALPC        = $06;    ! word
Constant HDR_DICTIONARY       = $08;    ! word
Constant HDR_OBJECTS          = $0A;    ! word
Constant HDR_GLOBALS          = $0C;    ! word
Constant HDR_STATICMEMORY     = $0E;    ! word
Constant HDR_GAMEFLAGS        = $10;    ! word
Constant HDR_GAMESERIAL       = $12;    ! six ASCII characters
Constant HDR_ABBREVIATIONS    = $18;    ! word
Constant HDR_FILELENGTH       = $1A;    ! word
Constant HDR_CHECKSUM         = $1C;    ! word
Constant HDR_TERPNUMBER       = $1E;    ! byte
Constant HDR_TERPVERSION      = $1F;    ! byte
Constant HDR_SCREENHLINES     = $20;    ! byte
Constant HDR_SCREENWCHARS     = $21;    ! byte
Constant HDR_SCREENWUNITS     = $22;    ! word
Constant HDR_SCREENHUNITS     = $24;    ! word
Constant HDR_FONTWUNITS       = $26;    ! byte
Constant HDR_FONTHUNITS       = $27;    ! byte
Constant HDR_ROUTINEOFFSET     = $28;    ! word
Constant HDR_STRINGOFFSET     = $2A;    ! word
Constant HDR_BGCOLOUR         = $2C;    ! byte
Constant HDR_FGCOLOUR         = $2D;    ! byte

```

```

Constant HDR_TERMCHARS      = $2E;      ! word
Constant HDR_PIXELSTO3     = $30;      ! word
Constant HDR_TERPSTANDARD  = $32;      ! two bytes
Constant HDR_ALPHABET      = $34;      ! word
Constant HDR_EXTENSION     = $36;      ! word
Constant HDR_UNUSED        = $38;      ! two words
Constant HDR_INFORMVERSION = $3C;      ! four ASCII characters
#Endif;

```

§4. Glulx Definitions. We make similar header definitions for Glulx. Extensive further definitions, of constants needed to handle the Glk I/O layer, can be found in the “Infglk” section of “Glulx.i6t”; they are not used in the rest of the template layer, and would only get in the way here.

```

#IFDEF TARGET_GLULX;
Global unicode_gestalt_ok; ! Set if interpreter supports Unicode
! Offsets into Glulx header and start of ROM:
Constant HDR_MAGICNUMBER   = $00;      ! long word
Constant HDR_GLULXVERSION  = $04;      ! long word
Constant HDR_RAMSTART      = $08;      ! long word
Constant HDR_EXTSTART      = $0C;      ! long word
Constant HDR_ENDMEM        = $10;      ! long word
Constant HDR_STACKSIZE     = $14;      ! long word
Constant HDR_STARTFUNC     = $18;      ! long word
Constant HDR_DECODINGTBL   = $1C;      ! long word
Constant HDR_CHECKSUM      = $20;      ! long word
Constant ROM_INFO          = $24;      ! four ASCII characters
Constant ROM_MEMORYLAYOUT  = $28;      ! long word
Constant ROM_INFORMVERSION = $2C;      ! four ASCII characters
Constant ROM_COMPVERSION   = $30;      ! four ASCII characters
Constant ROM_GAMERELEASE   = $34;      ! short word
Constant ROM_GAMESERIAL    = $36;      ! six ASCII characters
#Endif;

```

§5. Powers of Two. I6 lacks support for logical shifts, and the Z-machine opcodes which bear on this are not always well supported, so the I6 library has traditionally used a lookup table for the values of 2^{15-n} where $0 \leq n \leq 11$.

```

Array PowersOfTwo_TB
--> $$100000000000
    $$010000000000
    $$001000000000
    $$000100000000
    $$000010000000
    $$000001000000
    $$000000100000
    $$000000010000
    $$000000001000
    $$000000000100
    $$000000000010
    $$000000000001;
Array IncreasingPowersOfTwo_TB

```

```

--> $$00000000000000001
    $$00000000000000010
    $$00000000000000100
    $$00000000000001000
    $$00000000000010000
    $$00000000000100000
    $$00000000001000000
    $$00000000010000000
    $$00000000100000000
    $$00000001000000000
    $$00000001000000000
    $$00000010000000000
    $$00000100000000000
    $$00001000000000000
    $$00010000000000000
    $$00100000000000000
    $$01000000000000000
    $$10000000000000000;

```

§6. **Text Styles.** These are the styles of text distinguished by the template layer, though they are not required to look different from each other on any given VM. The codes are independent of the VM targetted, though in fact they are equal to Glulx style numbers as conventionally used. (The Z-machine renders some as roman, some as bold, but for instance makes `HEADER_VMSTY` and `SUBHEADER_VMSTY` indistinguishable to the eye.) Glulx's system of styles is one of its weakest points from an IF author's perspective, since it is all but impossible to achieve the text effects one might want – boldface, for instance, or red text – and text rendering is almost the only area in which it is clearly inferior to the Z-machine, which it was designed to replace. Still, using these styles when we can will get the most out of it, and for unornamental works Glulx is fine in practice.

```

Constant NORMAL_VMSTY      = 0;
Constant HEADER_VMSTY     = 3;
Constant SUBHEADER_VMSTY  = 4;
Constant ALERT_VMSTY     = 5;
Constant NOTE_VMSTY      = 6;
Constant BLOCKQUOTE_VMSTY = 7;
Constant INPUT_VMSTY     = 8;

```

§7. **Colour Numbers.** These are traditional colour names: quite who it was who thought that azure was the same colour as cyan is now unclear. Colour is, again, not easy to arrange on Glulx, but there is some workaround code.

```

Constant CLR_DEFAULT = 1;
Constant CLR_BLACK   = 2;
Constant CLR_RED     = 3;
Constant CLR_GREEN   = 4;
Constant CLR_YELLOW  = 5;
Constant CLR_BLUE    = 6;
Constant CLR_MAGENTA = 7; Constant CLR_PURPLE = 7;
Constant CLR_CYAN   = 8; Constant CLR_AZURE  = 8;
Constant CLR_WHITE  = 9;

```

§8. Window Numbers. Although Glulx can support elaborate tessalations of windows on screen (if the complexity of handling this can be mastered), the Z-machine has much more limited resources in general, so the template layer assumes a simple screen model: there are just two screen areas, the scrolling main window in which commands are typed and responses printed, and the fixed status line bar at the top of the screen.

```
Constant WIN_ALL      = 0; ! Both windows at once
Constant WIN_STATUS  = 1;
Constant WIN_MAIN    = 2;
```

§9. Paragraphing Flags. I am not sure many dictionaries would countenance “to paragraph” as a verb, but never mind: the reference here is to the algorithm used to place paragraph breaks within text, which uses bitmaps composed of the following.

```
Constant PARA_COMPLETED      = 1;
Constant PARA_PROMPTSKIP    = 2;
Constant PARA_SUPPRESSPROMPTSKIP = 4;
Constant PARA_NORULEBOOKBREAKS = 8;
Constant PARA_CONTENTEXPECTED = 16;
```

§10. Descriptors in the Language of Play. The following constants, which must be different in value from the number of any I6 attribute, are used in the `LanguageDescriptors` table found in the definition of the language of play.

```
Constant POSSESS_PK = $100;
Constant DEFART_PK  = $101;
Constant INDEFART_PK = $102;
```

§11. Run-Time Problem Numbers. The enumeration sequence here must correspond to that in the file of RTP texts, which is used to generate the HTML pages displayed by the Inform user interface when a run-time problem has occurred. (For instance, the file of RTP texts includes the heading “P17 - Can’t divide by zero”, equivalent to `RTP_DIVZERO` being 17 below.) There is no significance to the sequence, which is simply the historical order in which they were added to I7.

```
Constant RTP_BACKDROP          = 1;
Constant RTP_EXITDOOR         = 2;
Constant RTP_NOEXIT           = 3;
Constant RTP_CANTCHANGE       = 4;
Constant RTP_IMPREL           = 5;
Constant RTP_RULESTACK        = 6;
Constant RTP_TOOMANYRULEBOOKS = 7;
Constant RTP_TOOMANYEVENTS    = 8;
Constant RTP_BADPROPERTY      = 9;
Constant RTP_UNPROVIDED       = 10;
Constant RTP_UNSET            = 11;
Constant RTP_TOOMANYACTS      = 12;
Constant RTP_CANTABANDON      = 13;
Constant RTP_CANTEND          = 14;
Constant RTP_CANTMOVENOTHING  = 15;
Constant RTP_CANTREMOVENOTHING = 16;
Constant RTP_DIVZERO          = 17;
Constant RTP_BADVALUEPROPERTY = 18;
Constant RTP_NOTBACKDROP      = 19;
```

```
Constant RTP_TABLE_NOCOL           = 20;
Constant RTP_TABLE_NOCORR          = 21;
Constant RTP_TABLE_NOROW           = 22;
Constant RTP_TABLE_NOENTRY         = 23;
Constant RTP_TABLE_NOTABLE         = 24;
Constant RTP_TABLE_NOMOREBLANKS    = 25;
Constant RTP_TABLE_NOROWS          = 26;
Constant RTP_TABLE_CANTSORT        = 27;
Constant RTP_NOTINAROOM            = 28;
Constant RTP_BADTOPIC              = 29;
Constant RTP_ROUTELESS             = 30;
Constant RTP_PROPOFNOTHING         = 31;
Constant RTP_DECIDEONWRONGKIND     = 32;
Constant RTP_DECIDEONNOTHING       = 33;
Constant RTP_TABLE_CANTSAVE        = 34;
Constant RTP_TABLE_WONTFIT         = 35;
Constant RTP_TABLE_BADFILE         = 36;
Constant RTP_LOWLEVELERROR         = 37;
Constant RTP_DONTIGNORETURNSEQUENCE = 38;
Constant RTP_SAYINVALIDSNIPPET    = 39;
Constant RTP_SPLICEINVALIDSNIPPET = 40;
Constant RTP_INCLUDEINVALIDSNIPPET = 41;
Constant RTP_LISTWRITERMEMORY      = 42;
Constant RTP_CANTREMOVEPLAYER      = 43;
Constant RTP_CANTREMOVEDOORS       = 44;
Constant RTP_CANTCHANGEOFFSTAGE    = 45;
Constant RTP_MSTACKMEMORY          = 46;
Constant RTP_TYPECHECK             = 47;
Constant RTP_FILEIOERROR           = 48;
Constant RTP_HEAPERROR             = 49;
Constant RTP_LISTRANGEERROR        = 50;
Constant RTP_REGEXPSYNTAXERROR     = 51;
Constant RTP_NOGLULXUNICODE        = 52;
Constant RTP_BACKDROPONLY          = 53;
Constant RTP_NOTHING               = 54;
Constant RTP_SCENEHASNTSTARTED     = 55;
Constant RTP_SCENEHASNTENDED       = 56;
Constant RTP_NEGATIVEROOT          = 57;
Constant RTP_TABLE_CANTRUNTHROUGH  = 58;
Constant RTP_CANTITERATE           = 59;
```

§12. Template Activities. These are the activities defined in the Standard Rules. Most, though not all, are carried out by explicit function calls in the template layer, which is why we need their ID numbers: note that NI assigns each activity a unique ID number on creation, counting upwards from 0, and that it processes the Standard Rules before any other source text. (These numbers must correspond *both* to those in the source of NI, *and* to the creation sequence in the Standard Rules.)

```

Constant PRINTING_THE_NAME_ACT          = 0;
Constant PRINTING_THE_PLURAL_NAME_ACT   = 1;
Constant PRINTING_A_NUMBER_OF_ACT       = 2;
Constant PRINTING_ROOM_DESC_DETAILS_ACT = 3;
Constant LISTING_CONTENTS_ACT           = 4;
Constant GROUPING_TOGETHER_ACT          = 5;
Constant WRITING_A_PARAGRAPH_ABOUT_ACT  = 6;
Constant LISTING_NONDESCRIPT_ITEMS_ACT  = 7;

Constant PRINTING_NAME_OF_DARK_ROOM_ACT = 8;
Constant PRINTING_DESC_OF_DARK_ROOM_ACT = 9;
Constant PRINTING_NEWS_OF_DARKNESS_ACT  = 10;
Constant PRINTING_NEWS_OF_LIGHT_ACT     = 11;
Constant REFUSAL_TO_ACT_IN_DARK_ACT     = 12;

Constant CONSTRUCTING_STATUS_LINE_ACT    = 13;
Constant PRINTING_BANNER_TEXT_ACT        = 14;

Constant READING_A_COMMAND_ACT           = 15;
Constant DECIDING_SCOPE_ACT              = 16;
Constant DECIDING_CONCEALED_POSSESS_ACT  = 17;
Constant DECIDING_WHETHER_ALL_INC_ACT    = 18;
Constant CLARIFYING_PARSERS_CHOICE_ACT   = 19;
Constant ASKING_WHICH_DO_YOU_MEAN_ACT    = 20;
Constant PRINTING_A_PARSER_ERROR_ACT     = 21;
Constant SUPPLYING_A_MISSING_NOUN_ACT    = 22;
Constant SUPPLYING_A_MISSING_SECOND_ACT   = 23;
Constant IMPLICITLY_TAKING_ACT           = 24;
Constant STARTING_VIRTUAL_MACHINE_ACT    = 25;

Constant AMUSING_A_VICTORIOUS_PLAYER_ACT = 26;
Constant PRINTING_PLAYERS_OBITUARY_ACT   = 27;
Constant DEALING_WITH_FINAL_QUESTION_ACT = 28;

Constant PRINTING_LOCALE_DESCRIPTION_ACT  = 29;
Constant CHOOSING_NOTABLE_LOCALE_OBJ_ACT = 30;
Constant PRINTING_LOCALE_PARAGRAPH_ACT   = 31;

```

§13. **Template Rulebooks.** Rulebooks are created in a similar way, and again are numbered upwards from 0 in order of creation. These are the ones used in the template layer. (These numbers must correspond *both* to those in the source of NI, *and* to the creation sequence in the Standard Rules.)

```
Constant PROCEDURAL_RB      = 0;
Constant STARTUP_RB        = 1;
Constant TURN_SEQUENCE_RB  = 2;
Constant SHUTDOWN_RB       = 3;
Constant WHEN_PLAY_BEGINS_RB = 5;
Constant WHEN_PLAY_ENDS_RB  = 6;
Constant WHEN_SCENE_BEGINS_RB = 7;
Constant WHEN_SCENE_ENDS_RB = 8;
Constant ACTION_PROCESSING_RB = 10;
Constant SETTING_ACTION_VARIABLES_RB = 11;
Constant SPECIFIC_ACTION_PROCESSING_RB = 12;
Constant ACCESSIBILITY_RB   = 14;
Constant REACHING_INSIDE_RB  = 15;
Constant REACHING_OUTSIDE_RB = 16;
Constant VISIBLE_RB         = 17;
Constant PERSUADE_RB        = 18;
Constant UNSUCCESSFUL_ATTEMPT_RB = 19;
Constant AFTER_RB          = 24;
Constant REPORT_RB         = 25;
```

§14. **Data Type IDs.** These are filled in automatically by NI, and have the same names as are used in the NI source (and in the Types.i6t section): for instance NUMBER_TY.

```
{-call:Types::IDs::compile_I6_constants}
```

§15. **Parser Error Numbers.** The traditional ways in which the I6 library's parser, which we adopt here more or less intact, can give up on a player's command. See the *Inform Designer's Manual*, 4th edition, for details.

```
Constant STUCK_PE      = 1;
Constant UPTO_PE      = 2;
Constant NUMBER_PE    = 3;
Constant ANIMA_PE     = 4;
Constant CANTSEE_PE   = 5;
Constant TOOLIT_PE    = 6;
Constant NOTHELD_PE   = 7;
Constant MULTI_PE     = 8;
Constant MMULTI_PE    = 9;
Constant VAGUE_PE     = 10;
Constant EXCEPT_PE  = 11;
Constant VERB_PE      = 12;
Constant SCENERY_PE   = 13;
Constant ITGONE_PE    = 14;
Constant JUNKAFTER_PE = 15;
Constant TOOFEW_PE    = 16;
Constant NOTHING_PE   = 17;
Constant ASKSCOPE_PE  = 18;
Constant NOTINCONTEXT_PE = 19;
Constant BLANKLINE_PE = 20; ! Not formally a parser error, but used by I7 as if
```

§16. Scope Searching Reasons. The parser makes use of a mechanism for searching through the objects currently “in scope”, which basically means visible to the actor who would perform the action envisaged by the command being parsed. It is sometimes useful to behave differently depending on why this scope searching is being done, so the following constants enumerate the possibilities.

I6’s `EACH_TURN_REASON`, `REACT_BEFORE_REASON` and `REACT_AFTER_REASON` have been removed from this list as no longer meaningful; hence the lacuna in numbering.

```
Constant PARSING_REASON      = 0;
Constant TALKING_REASON      = 1;
Constant EACH_TURN_REASON    = 2;
Constant LOOPOVERSCOPE_REASON = 5;
Constant TESTSCOPE_REASON    = 6;
```

§17. Token Types. Tokens are the indecomposable pieces of a grammar line making up a possible reading of a command; some are literal words, others stand for “any named object in scope”, and so on. The following codes identify the possibilities. The `*_TOKEN` constants must not be altered without modifying the I6 compiler to match (so, basically, they must not be altered at all).

```
Constant ILLEGAL_TT          = 0;    ! Types of grammar token: illegal
Constant ELEMENTARY_TT       = 1;    !      (one of those below)
Constant PREPOSITION_TT      = 2;    !      e.g. 'into'
Constant ROUTINE_FILTER_TT   = 3;    !      e.g. noun=CagedCreature
Constant ATTR_FILTER_TT      = 4;    !      e.g. edible
Constant SCOPE_TT            = 5;    !      e.g. scope=Spells
Constant GPR_TT              = 6;    !      a general parsing routine

Constant NOUN_TOKEN          = 0;    ! The elementary grammar tokens, and
Constant HELD_TOKEN          = 1;    ! the numbers compiled by I6 to
Constant MULTI_TOKEN         = 2;    ! encode them
Constant MULTIHELD_TOKEN     = 3;
Constant MULTIEXCEPT_TOKEN = 4;
Constant MULTIINSIDE_TOKEN   = 5;
Constant CREATURE_TOKEN      = 6;
Constant SPECIAL_TOKEN       = 7;
Constant NUMBER_TOKEN        = 8;
Constant TOPIC_TOKEN         = 9;
Constant ENDIT_TOKEN         = 15;   ! Value used to mean "end of grammar line"
```

§18. **GPR Return Values.** GRP stands for “General Parsing Routine”, an I6 routine which acts as a grammar token: again, see the *Inform Designer’s Manual*, 4th edition, for details.

In Library 6/11, GPR_NOUN is defined as \$ff00, but this would fail on Glulx: it needs to be \$ffffff00 on 32-bit virtual machines. It appears that GPR_NOUN to GPR_CREATURE, though documented in the old *Inform Translator’s Manual*, were omitted when this was consolidated into the DM4, so that they effectively disappeared from view. But they might still be useful for implementing inflected forms of nouns, so we have retained them here regardless.

```
Constant GPR_FAIL          = -1;    ! Return values from General Parsing
Constant GPR_PREPOSITION  = 0;     ! Routines
Constant GPR_NUMBER       = 1;
Constant GPR_MULTIPLE     = 2;
Constant GPR_REPARSE      = REPARSE_CODE;
Constant GPR_NOUN         = -256; ! Reparse, but as |NOUN_TOKEN| this time
Constant GPR_HELD        = GPR_NOUN + 1; ! And so on
Constant GPR_MULTI       = GPR_NOUN + 2;
Constant GPR_MULTIHELD   = GPR_NOUN + 3;
Constant GPR_MULTIEXCEPT = GPR_NOUN + 4;
Constant GPR_MULTIIINSIDE = GPR_NOUN + 5;
Constant GPR_CREATURE     = GPR_NOUN + 6;
```

§19. **List Styles.** These constants make up bitmaps of the options in use when listing objects.

```
Constant NEWLINE_BIT      = $$0000000000000001; ! New-line after each entry
Constant INDENT_BIT       = $$0000000000000010; ! Indent each entry by depth
Constant FULLINV_BIT     = $$0000000000000100; ! Full inventory information after entry
Constant ENGLISH_BIT     = $$0000000000001000; ! English sentence style, with commas and and
Constant RECURSE_BIT     = $$0000000000010000; ! Recurse downwards with usual rules
Constant ALWAYS_BIT      = $$0000000001000000; ! Always recurse downwards
Constant TERSE_BIT       = $$0000000001000000; ! More terse English style
Constant PARTINV_BIT     = $$0000000010000000; ! Only brief inventory information after entry
Constant DEFART_BIT      = $$0000000100000000; ! Use the definite article in list
Constant WORKFLAG_BIT    = $$0000001000000000; ! At top level (only), only list objects
                                ! which have the "workflag" attribute
Constant ISARE_BIT       = $$0000010000000000; ! Print " is" or " are" before list
Constant CONCEAL_BIT     = $$0000100000000000; ! Omit objects with "concealed" or "scenery":
                                ! if WORKFLAG_BIT also set, then does not
                                ! apply at top level, but does lower down
Constant NOARTICLE_BIT   = $$0001000000000000; ! Print no articles, definite or not
Constant EXTRAINDENT_BIT = $$0010000000000000; ! New in I7: extra indentation of 1 level
Constant CFIRSTART_BIT   = $$0100000000000000; ! Capitalise first article in list
```

§20. **Lengths Of Time.** Inform measures time in minutes.

```
Constant QUARTER_HOUR = 15;
Constant HALF_HOUR   = 30;
Constant ONE_HOUR     = 60;
Constant TWELVE_HOURS = 720;
Constant TWENTY_FOUR_HOURS = 1440;
```

§21. **Empty Text.** The I6 compiler does not optimise string compilation: if it needs to compile the (packed, read-only) string "exemplum" twice, it will compile two copies. This is slightly wasteful on memory, though in practice the loss is not enough for us to care. But we do want to avoid this in I7 because, to make string-sorting algorithms more efficient, we want direct numerical comparison of packed addresses to be equivalent to string comparison: and that means the text "exemplum" has to be compiled once and once only. There's a general mechanism for this in NI, but the single case most often needed is the empty text, since this is the default value for text variables and properties: we give it a name as follows.

(This works because I6 constant definition is not like C preprocessor macro expansion: `EMPTY_TEXT_VALUE` is equated with the address resulting from compiling "", rather than being replaced with the blank text to be recompiled each time.)

```
Constant EMPTY_TEXT_VALUE "";
```

§22. **Empty Table.** Similarly: the default value for the "table" kind of value, a Table containing no rows and no columns.

```
Array TheEmptyTable --> 0 0;
```

§23. **Empty Rulebook.** Similarly. An empty rulebook is one whose array has first word equal to `NULL`: we define a sequence of four `$ff` bytes so that the first word will be `NULL` on either 16-bit or 32-bit VMs. As with the empty text, this array is the value of many empty rulebooks: rulebook arrays are read-only, so there is no risk of a clash.

```
Array EMPTY_RULEBOOK -> $ff $ff $ff $ff;
```

§24. **Empty Set.** The falsity proposition describes the empty set of objects, and is the zero value for the "description" kind of value.

```
[ Prop_Falsity reason obj; return 0; ];
```

§25. **Score and Rankings Table.** The following command tells NI to compile constant definitions for `MAX_SCORE` and/or `RANKING_TABLE`, in cases where there are scores and rankings. If there's no score, we define `MAX_SCORE` as 0 anyway; if there's no ranking table, `RANKING_TABLE` is left undefined, so that we can `#ifdef` this possibility later.

```
{-call:Data::Tables::compile_max_score}
#ifdef MAX_SCORE;
Global MAX_SCORE = 0;
#endif;
```

§26. Template Attributes. An I6 attribute is equivalent to an I7 “either/or property”, though the latter are not always translated into I6 attributes because the Z-machine has only a limited number of attributes to use. Here, we define attributes used by the template.

Many concepts in I6 correspond directly to their successors in I7, even if details may vary. (Value properties are a case in point.) Attributes are the opposite of this: indeed, no I6 concept is more fragmented in its I7 equivalents. All but one of the old I6 library attributes are still used (the **general** attribute, for miscellaneous use, has been removed: it more often invited abuse than use); and a few new attributes have been added. But they are used for a variety of purposes. Some do correspond exactly to either/or properties in I7, but others are a sort of signature for I7 kinds. (So that for I7 use they are read-only.) Others still are used by the template layer as part of the implementation of services for I7, but are not visible to I7 source text as storage.

```

Attribute absent; ! Used to mark objects removed from play
Attribute animate; ! I6-level marker for I7 kind "person"
Attribute clothing; ! = I7 "wearable"
Attribute concealed; ! = I7 "undescribed"
Attribute container; ! I6-level marker for I7 kind "container"
Attribute door; ! I6-level marker for I7 kind "door"
Attribute edible; ! = I7 "edible" vs "inedible"
Attribute enterable; ! = I7 "enterable"
Attribute light; ! = I7 "lighted" vs "dark"
Attribute lockable; ! = I7 "lockable"
Attribute locked; ! = I7 "locked"
Attribute moved; ! = I7 "handled"
Attribute on; ! = I7 "switched on" vs "switched off"
Attribute open; ! = I7 "open" vs "closed"
Attribute openable; ! = I7 "openable"
Attribute scenery; ! = I7 "scenery"
Attribute static; ! = I7 "fixed in place" vs "portable"
Attribute supporter; ! I6-level marker for I7 kind "supporter"
Attribute switchable; ! I6-level marker for I7 kind "device"
Attribute talkable; ! Not currently used by I7, but retained for possible future use
Attribute transparent; ! = I7 "transparent" vs "opaque"
Attribute visited; ! = I7 "visited"
Attribute worn; ! marks that an object tree edge represents wearing

Attribute male; ! not directly used by I7, but available for languages with genders
Attribute female; ! = I7 "female" vs "male"
Attribute neuter; ! = I7 "neuter"
Attribute pluralname; ! = I7 "plural-named"
Attribute proper; ! = I7 "proper-named"
Attribute remove_proper; ! remember to remove proper again when using ChangePlayer next

Attribute privately_named; ! New in I7
Attribute mentioned; ! New in I7
Attribute pushable; ! New in I7

Attribute mark_as_room; ! Used in I7 to speed up testing "ofclass K1_room"
Attribute mark_as_thing; ! Used in I7 to speed up testing "ofclass K2_thing"

Attribute workflag; ! = I7 "marked for listing", but basically temporary workspace
Attribute workflag2; ! new in I7 and also temporary workspace
Constant list_filter_permits = privately_named; ! another I7 listwriter convenience

```

§27. Template Properties. As remarked above, these more often correspond to value properties in I7. To an experienced I6 user, though, the death toll of abolished I6 properties in I7 is breathtaking: in alphabetical order, `after`, `cant_go`, `daemon`, `each_turn`, `invent`, `life`, `number`, `orders`, `react_after`, `react_before`, `time_left`, `time_out`, `when_closed`, `when_off`, `when_on`, `when_open`. In May 2008, the direction properties `n_to`, `s_to`, `e_to`, ..., `out_to` joined the list of the missing.

The losses are numerous because of the shift from I6's object orientation to I7's rule orientation: information about the behaviour of objects is no longer thought of as data attached to them. At that, it could have been worse: a few unused I6 library properties have been retained for possible future use.

```
Property add_to_scope; ! used as in I6 to place component parts in scope
Property article "a"; ! used as in I6 to implement articles
Property capacity 100; ! = I7 "carrying capacity"
Property component_child; ! new in I7: forest structure holding "part of" relation
Property component_parent; ! new in I7
Property component_sibling; ! new in I7
Property description; ! = I7 "description"
Property door_dir; ! used to implement two-sided doors, but holds direction object, not a property
Property door_to; ! used as in I6 to implement two-sided doors
Property found_in; ! used as in I6 to implement two-sided doors and backdrops
Property initial; ! = I7 "initial description"
Property list_together; ! used as in I6 to implement "grouping together" activity
Property map_region; ! new in I7
Property parse_name 0; ! used as in I6 to implement "Understand... as..." grammars
Property plural; ! used as in I6 to implement plural names for duplicate objects
Property regional_found_in; ! new in I7
Property room_index; ! new in I7: storage for route-finding
Property short_name 0; ! = I7 "printed name"
Property vector; ! new in I7: storage for route-finding
Property with_key; ! = I7 "matching key"

Property IK_0; ! Instance count of the kind of the current object
Property IK_1; ! These are instance counts within kinds K1, K2, ...
Property IK_2; ! and it is efficient to declare the common ones with Property
Property IK_4; ! since this results in a slightly smaller story file
Property IK_5;
Property IK_6;
Property IK_8;

Property IK1_link; ! These are for linked lists used to make searches faster
Property IK2_link; ! and again it's memory-efficient to declare the common ones
Property IK5_link; !
Property IK6_link; !
Property IK8_link; !

Property articles; ! not used by I7, but an interesting hook in the parser
Property grammar; ! not used by I7, but an interesting hook in the parser
Property inside_description; ! not used by I7, but an interesting hook in the locale code
Property short_name_indef 0; ! not used by I7, but an interesting hook in the listmaker
```

§28. **Loss of Life.** The loss of `life` is so appalling that I6 will not even compile a story file which doesn't define the property number `life` (well, strictly speaking, it checks the presence of constants suggesting the I6 library first, but the template layer does define constants like that). We define it as a null constant to be sure of avoiding any valid property number; I6 being typeless, that enables the veneer to compile again. (The relevant code is in `CA_Pr`, defined in the `veneer.c` section of I6.)

```
Constant life = NULL;
```

§29. **Action Count.** The number of valid I7 actions in existence.

```
Constant ActionCount = {-value:NUMBER_CREATED(action_name)};
```

§30. **Fake Actions.** Though sometimes useful for I6 coding tricks, fake actions – action numbers not corresponding to any action, but different from those of valid actions, and useable with a number of action syntaxes – are not conceptually present in I7 source text. They can only really exist at the I6 level because I6 is typeless; in I7 terms, there is no convenient kind of value which could represent both actions and fake actions while protecting each from confusion with the other.

See the *Inform Designer's Manual*, 4th edition, for what these are used for.

The following fake actions from the I6 library have been dropped here: `##LetGo`, `##Receive`, `##ThrownAt`, `##Prompt`, `##Places`, `##Objects`, `##Order`, `##NotUnderstood`.

```
Fake_Action ListMiscellany;
Fake_Action Miscellany;
Fake_Action PluralFound;
Fake_Action TheSame;
```